

Calcolatori Elettronici T
Ingegneria Informatica

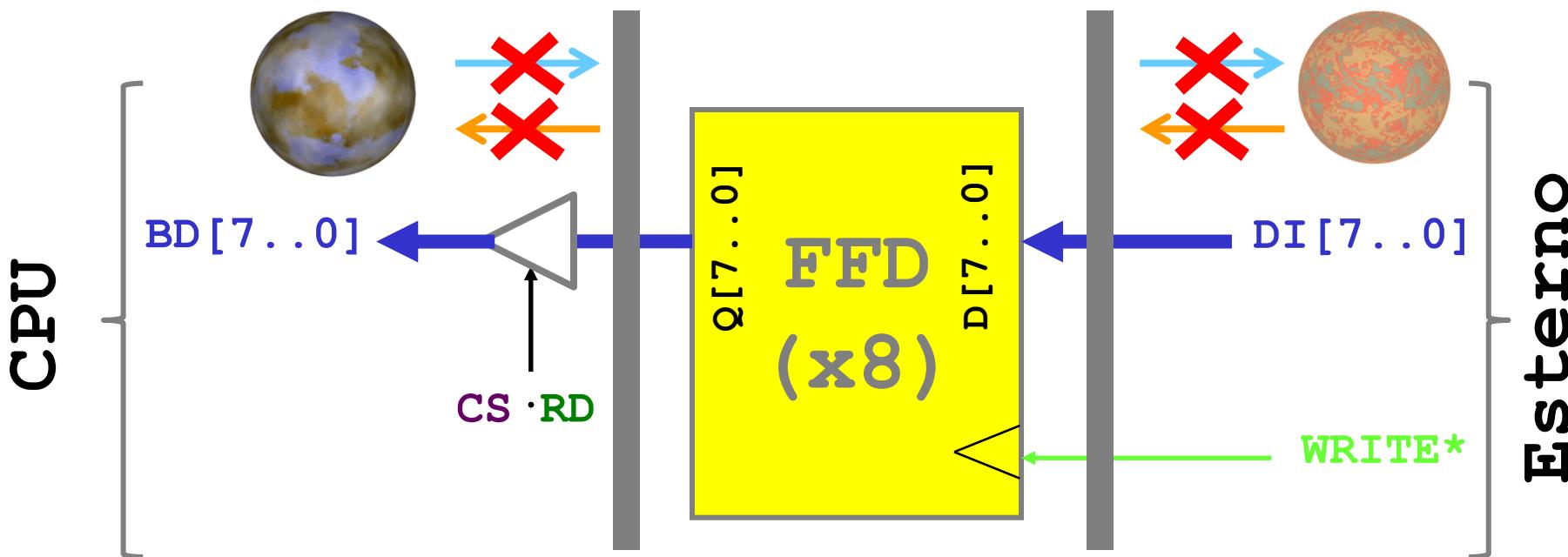
05 Periferiche di I/O con *handshake*



Porte di Input/Output (I/O)

- In precedenza abbiamo visto come progettare delle semplici periferiche di I/O, per scambiare dati tra CPU e mondo esterno mediante un *buffer*
- Tuttavia, non vi era nessuna garanzia sul corretto esito dei trasferimenti
- Infatti, cosa accade se, mentre la CPU scrive nella porta in output, un dispositivo esterno legge dalla medesima porta? Inoltre, cosa accade se la CPU legge un dato che in realtà non è mai stato scritto da un dispositivo esterno? Come può saperlo? Per questo, i trasferimenti sono intrinsecamente esposti a errori
- In più, la gestione del trasferimenti era totalmente a carico della CPU (che potrebbe fare altro)
- Le porte di I/O non erano in grado di generare interrupt con tutte le problematiche che ne derivano

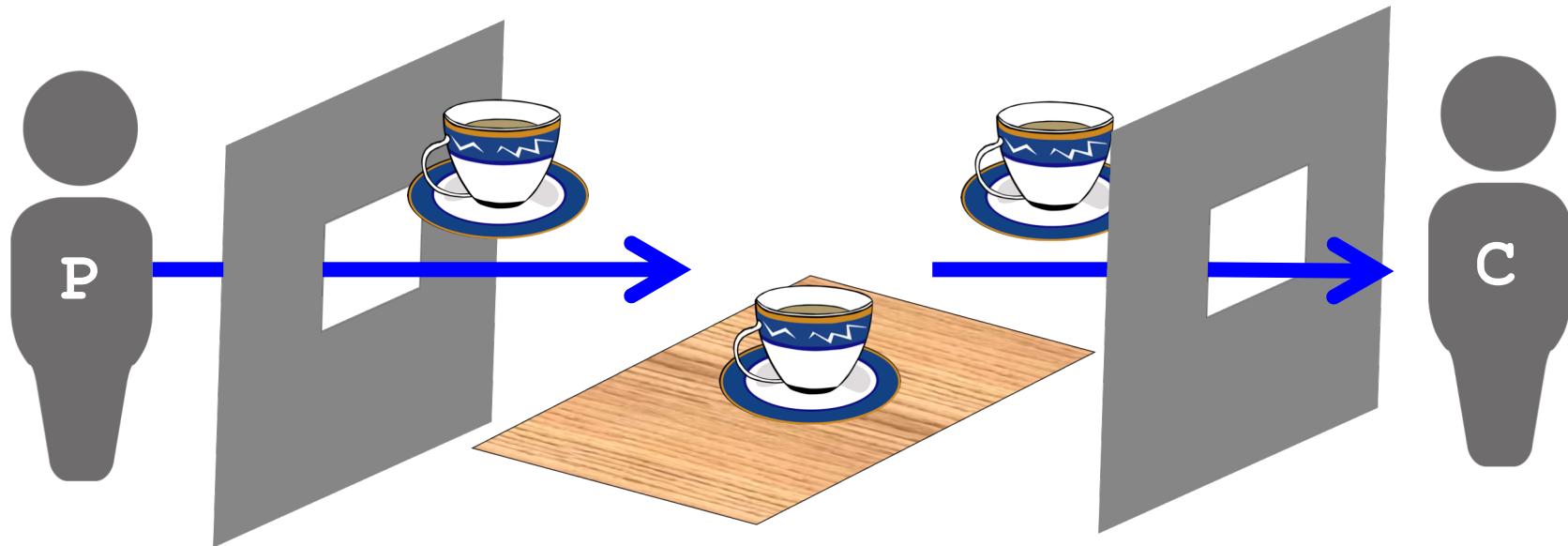
Una semplice periferica per leggere dati dall'esterno, senza utilizzare interrupt, è la seguente:



Tuttavia, con questa soluzione, sorgono degli evidenti problemi:

- Come può sapere la CPU che è disponibile un nuovo dato scritto dall'esterno nella porta?
- Come si può sapere dall'esterno che la CPU ha letto il dato scritto in precedenza nella porta?

Un esempio che evidenzia questi problemi riguarda lo scambio di beni/dati tra un produttore e un consumatore

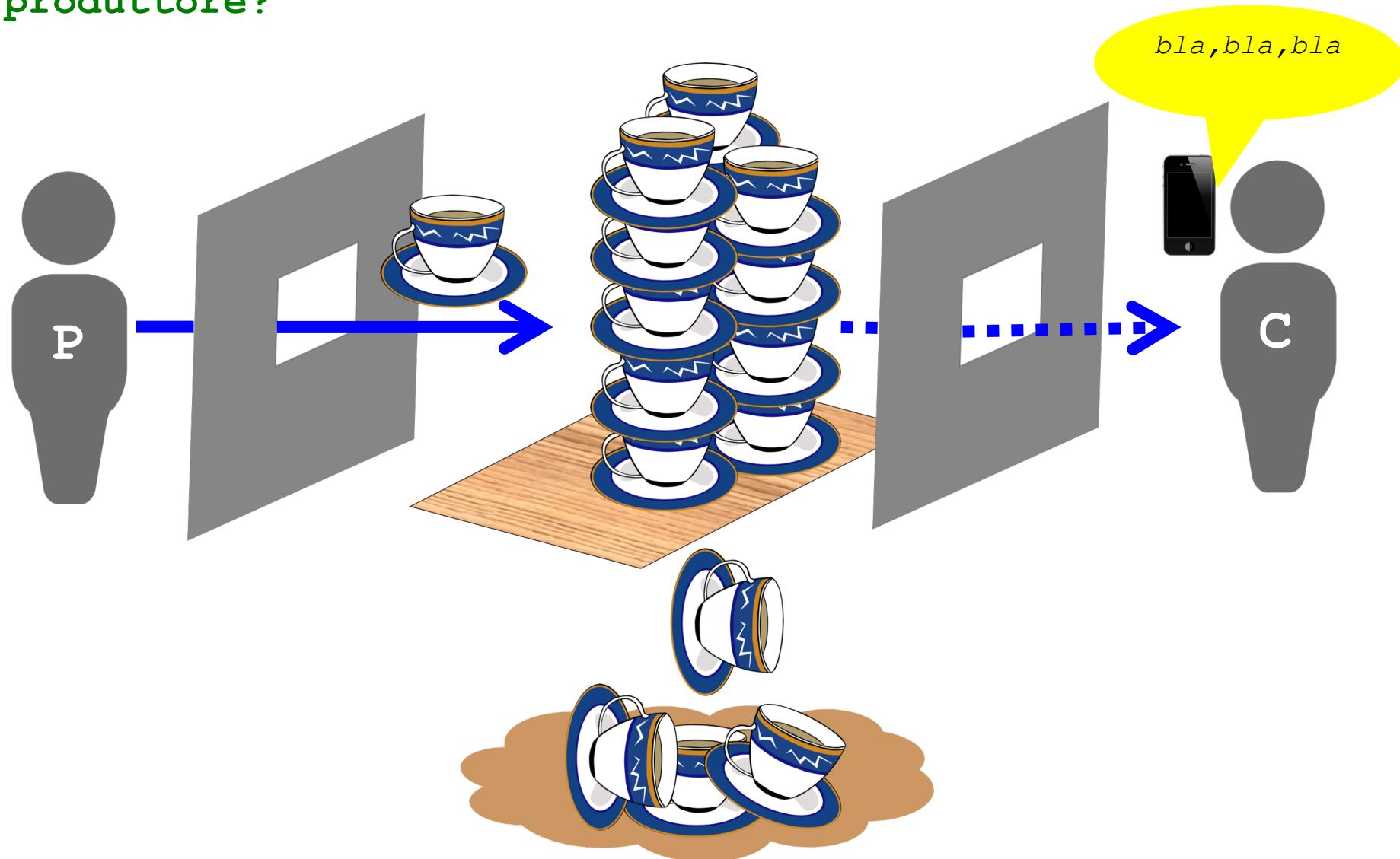


Se il produttore produce un caffè che è prelevato prima dell'arrivo di un altro tutto potrebbe apparentemente funzionare correttamente (**setup e hold?**)

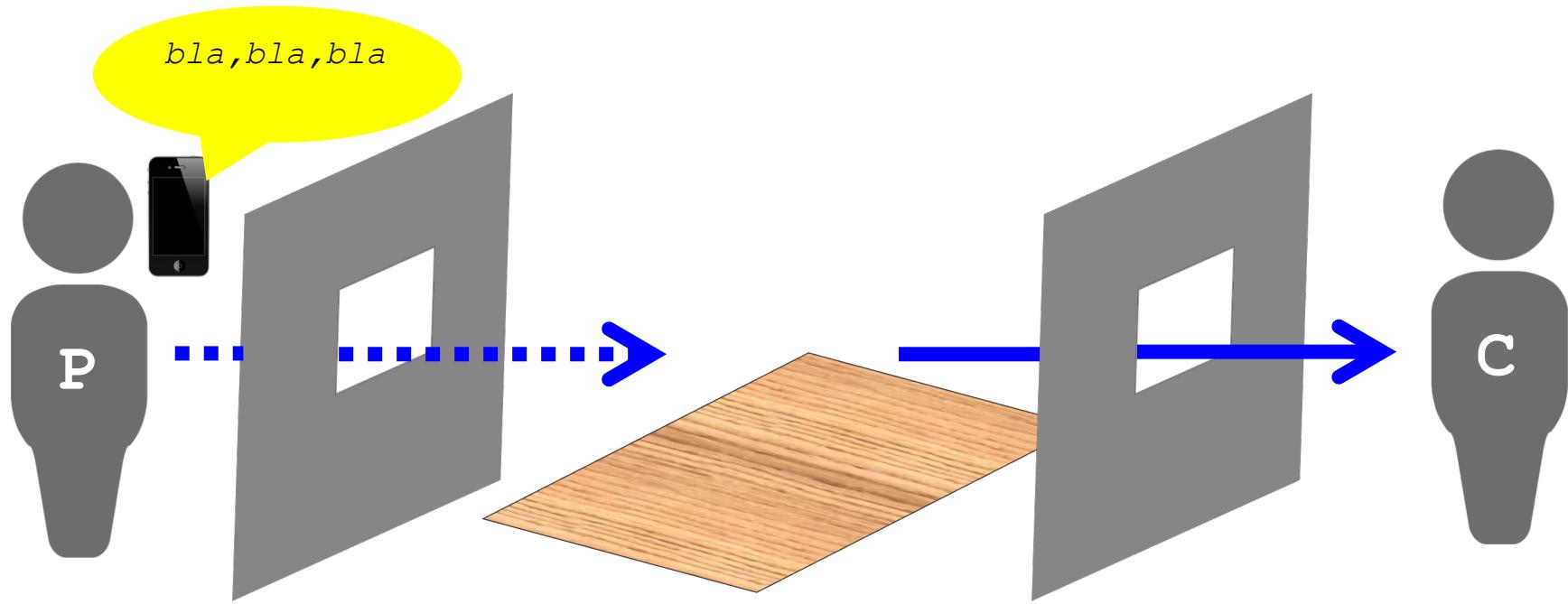
Tuttavia, come può sapere il produttore che il caffè è stato prelevato? Come può sapere il consumatore che è disponibile un nuovo caffè preparato dal produttore?

Per queste ragioni, sorgono altri problemi...

Un primo problema si verifica se il consumatore smette di prelevare caffè perché non è pronto (e.g., il consumatore è al telefono). Come può saperlo il produttore?



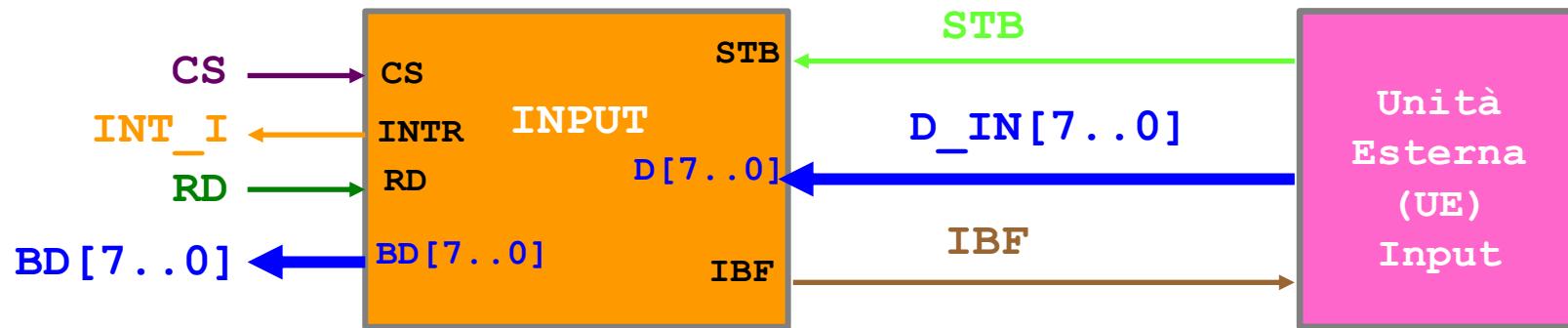
Il problema duale si verifica se il produttore smette di preparare caffè perché impegnato a fare altro (e.g., parlare al telefono). **Come può saperlo il consumatore?**



- I due problemi evidenziati possono essere risolti in modo molto semplice ricorrendo a qualche forma di **sincronizzazione** tra le due entità **P e C**
- Per questo scopo l'**handshake** è un approccio semplice, efficiente e ampiamente utilizzato

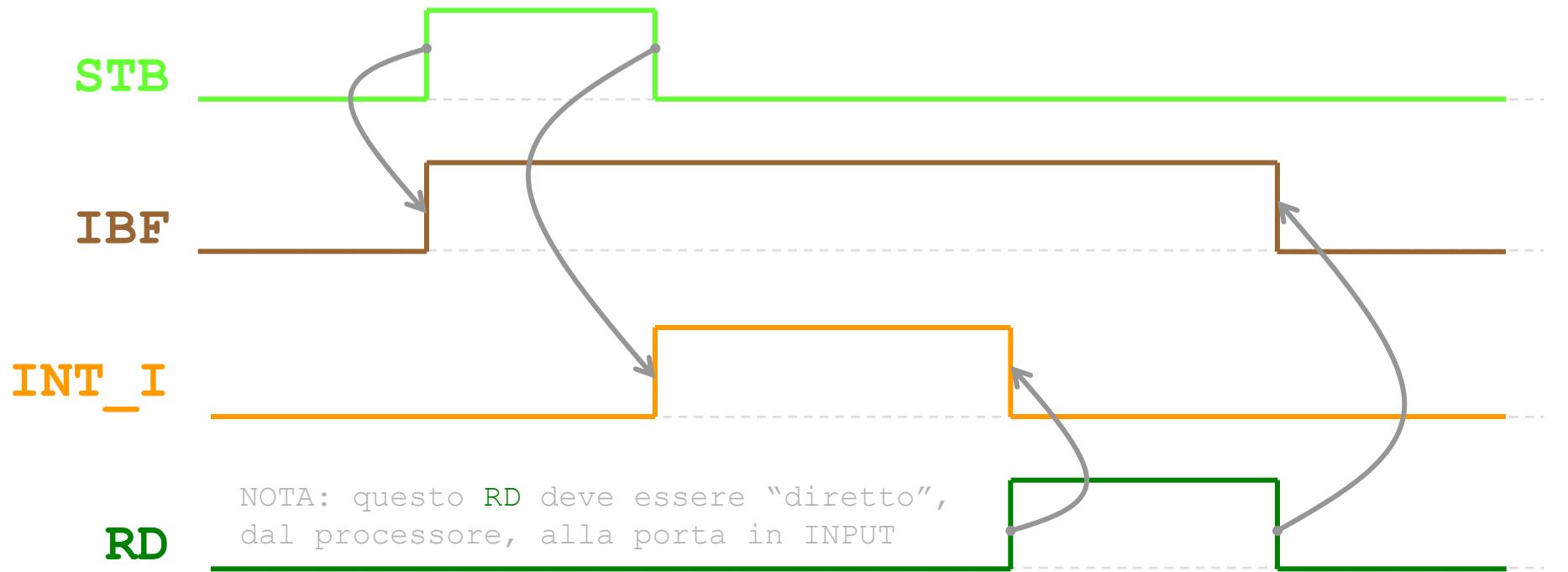
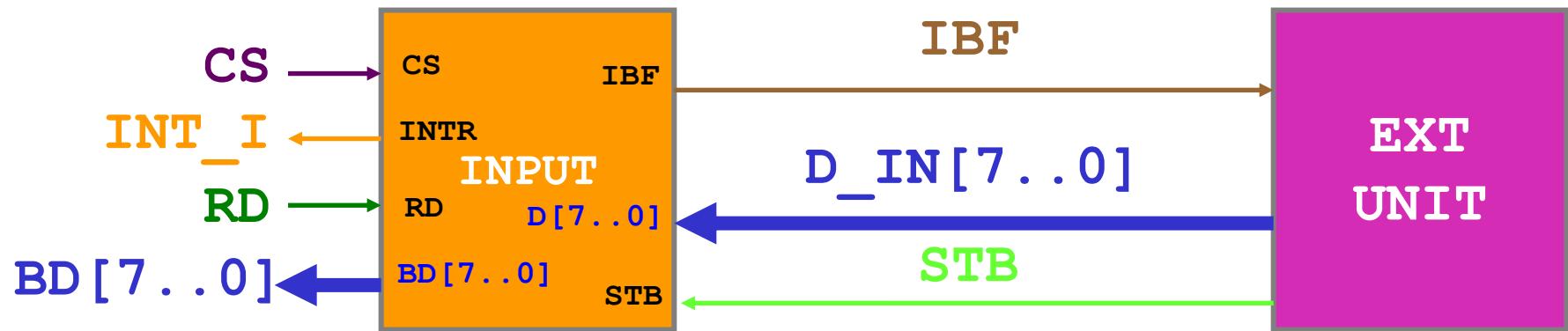


Segnali del protocollo “handshake”: INPUT



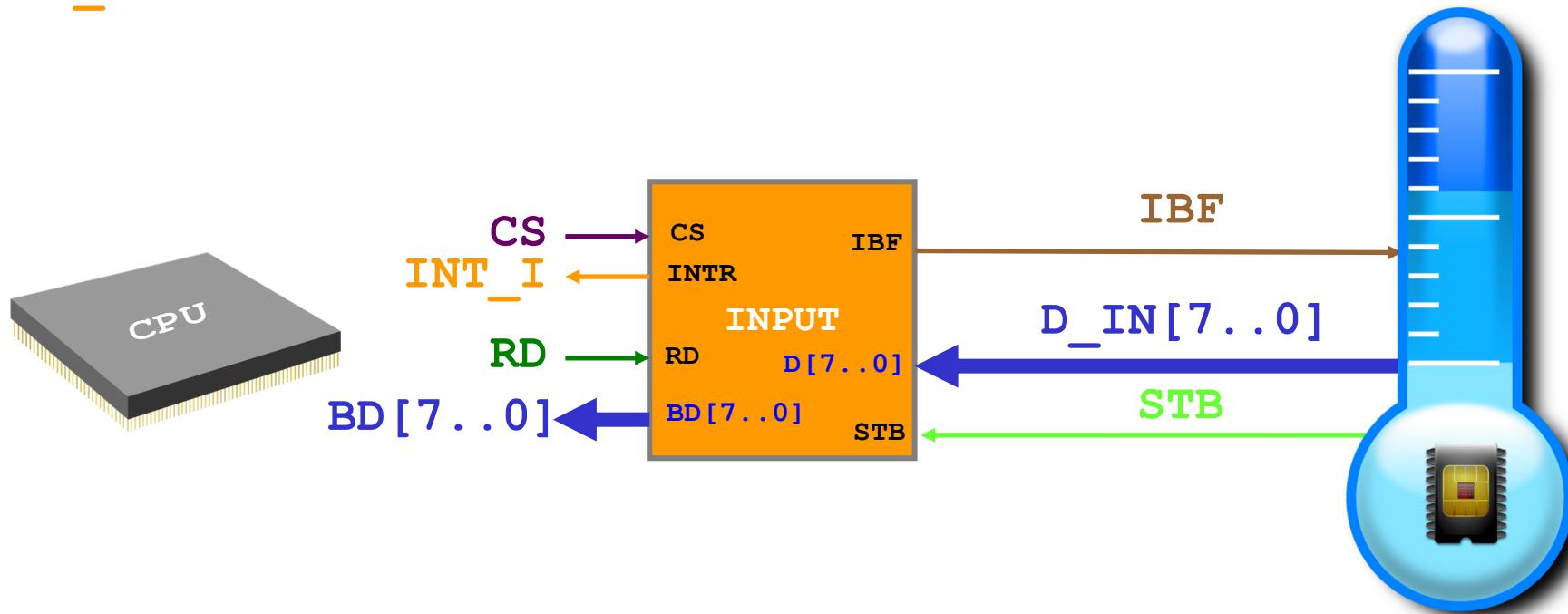
1. Se $IBF=0$, quando possibile* UE può scrivere il dato nel buffer d'ingresso della porta
2. UE, portando **STB** a 1, scrive il dato nella porta che contemporaneamente asserisce **IBF** (*Input Buffer Full*)
3. Quando UE porta **STB** a zero (scrittura terminata), l'interfaccia attiva **INT_I** (*Interrupt Request*)
4. Quando possibile*, la CPU andrà a leggere il dato scritto nella porta da UE. Al termine, **IBF** andrà a zero (mentre **INT_I** va a 0, dall'inizio della lettura)

Handshake (INPUT) : forme d'onda



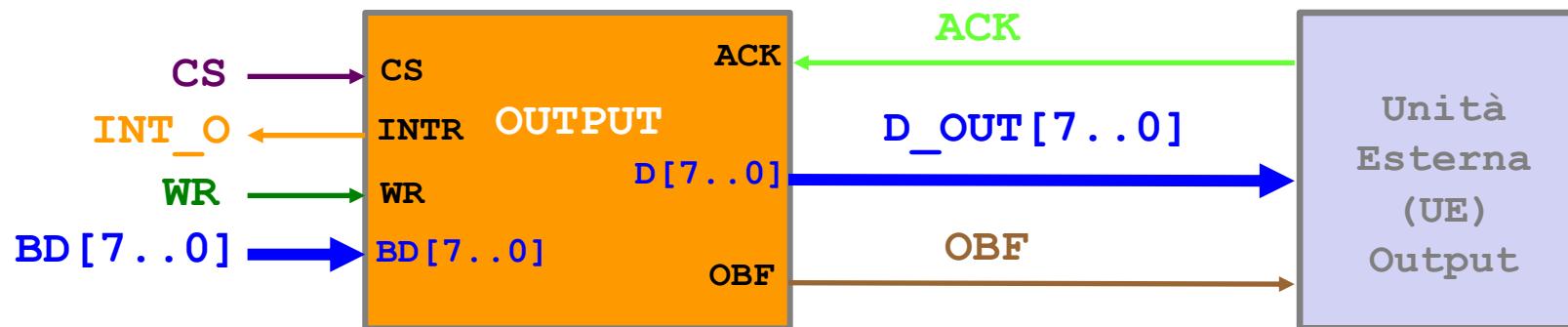
Un esempio di unità esterna in input potrebbe essere rappresentata da un sensore (e.g., di temperatura)

Il sensore invia i dati alla CPU attraverso la periferica di input quando una nuova misura è disponibile e **IBF=0**. Al termine di ogni scrittura nella porta da parte del sensore di temperatura, il segnale **INT_I** si asserisce



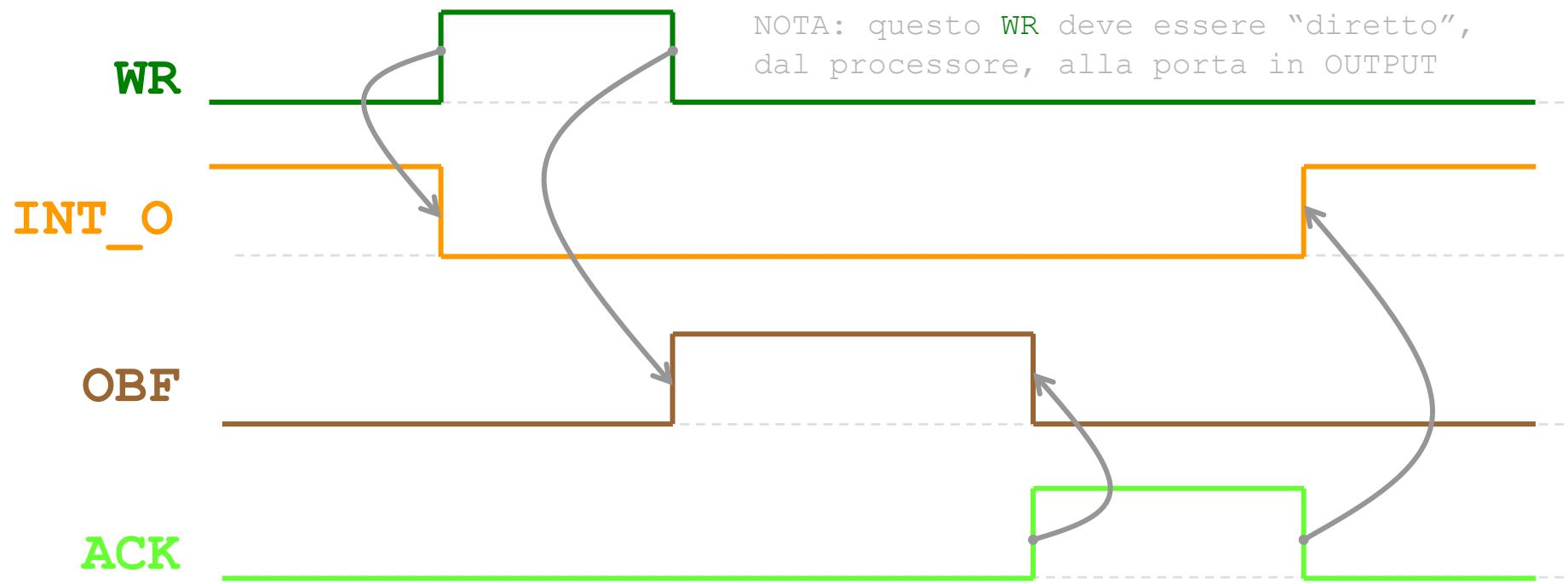
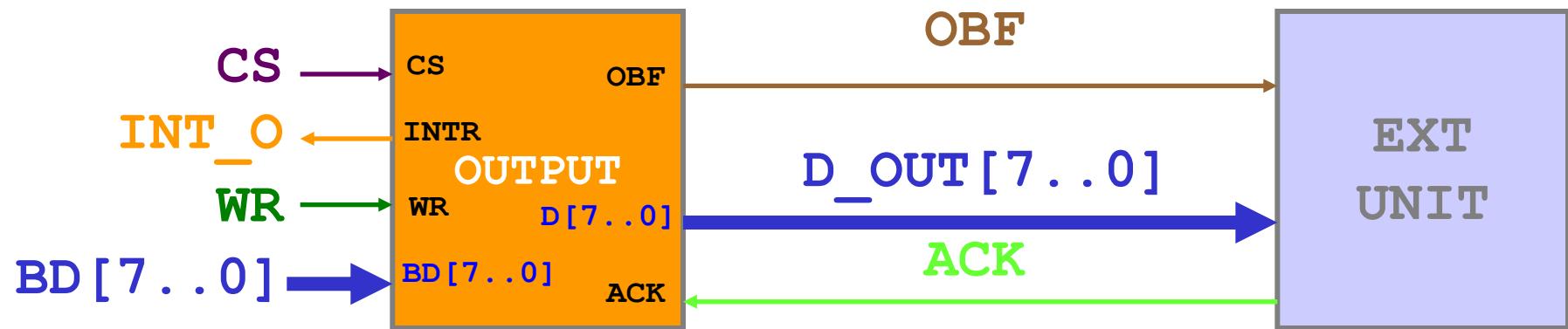
Il sensore deve contenere una semplice rete logica in grado di gestire il protocollo di handshake

Segnali del protocollo “handshake”: OUTPUT



1. Il segnale **INT_O** asserito comunica alla CPU che la porta può accettare un nuovo dato
2. In risposto alla richiesta di interrupt la CPU **scrive**, quando possibile*, il dato sul buffer della porta
1. L'interfaccia segnala a UE che è disponibile un nuovo dato attivando **OBF** (*Output Buffer Full*)
2. Quando possibile*, UE **legge** il dato scritto dalla CPU asserendo **ACK** (*acknowledge*)

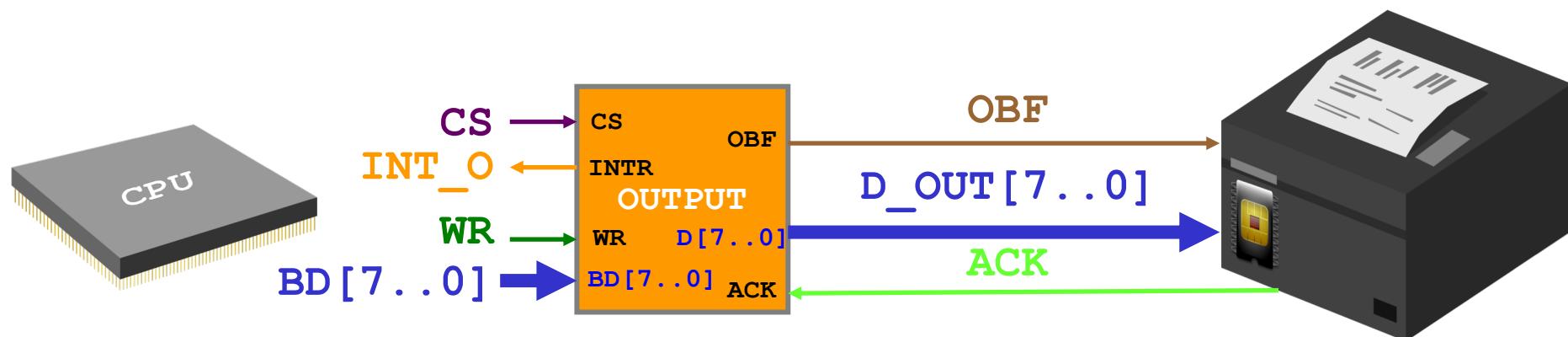
Handshake (OUTPUT) : forme d'onda



Un esempio di unità esterna in output potrebbe essere rappresentata da una stampante che imprime sulla carta un carattere alla volta.

La CPU, fornisce* i dati alla stampante attraverso la periferica di output quando il segnale **INT_O** è asserito (questo implica che **OBF** sia 0)

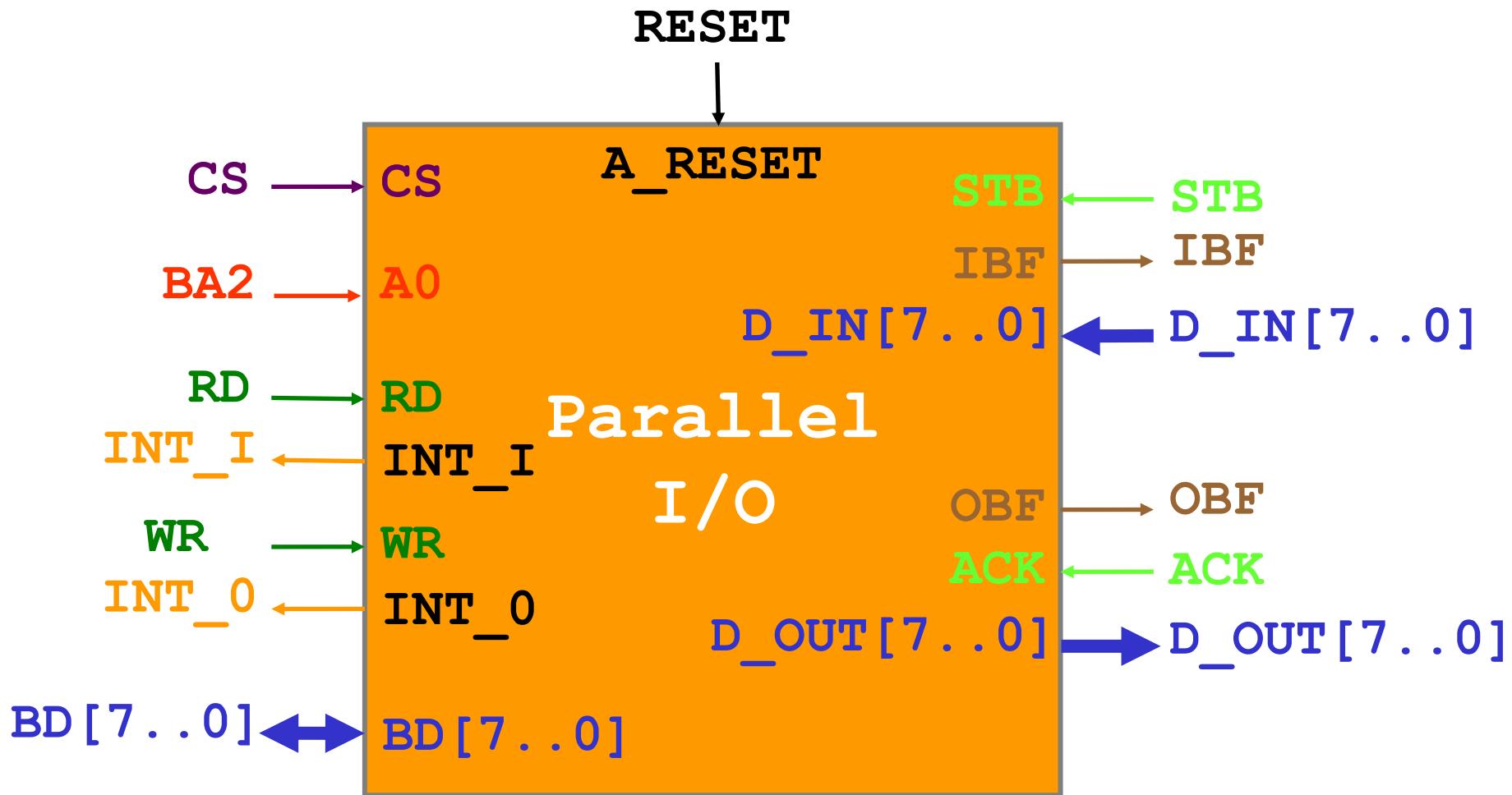
La stampante legge il dato solo quando il segnale **OBF** è asserito (i.e., quando la porta in output comunica alla stampante che un nuovo dato è stato scritto dalla CPU nel buffer ed è quindi disponibile)



La stampante deve contenere una semplice rete logica in grado di gestire il protocollo di handshake

Esercizio

Progettare una rete logica basata su FFD in grado di gestire le comunicazioni con due dispositivi esterni - uno in input mappato a **CS+0** e uno in output mappato a **CS+1** - utilizzando il protocollo di *handshake*



Progetto dell'interfaccia

L'interfaccia parallela è dotata di due porte, ciascuna in grado di trasferire dati a 8 bit:

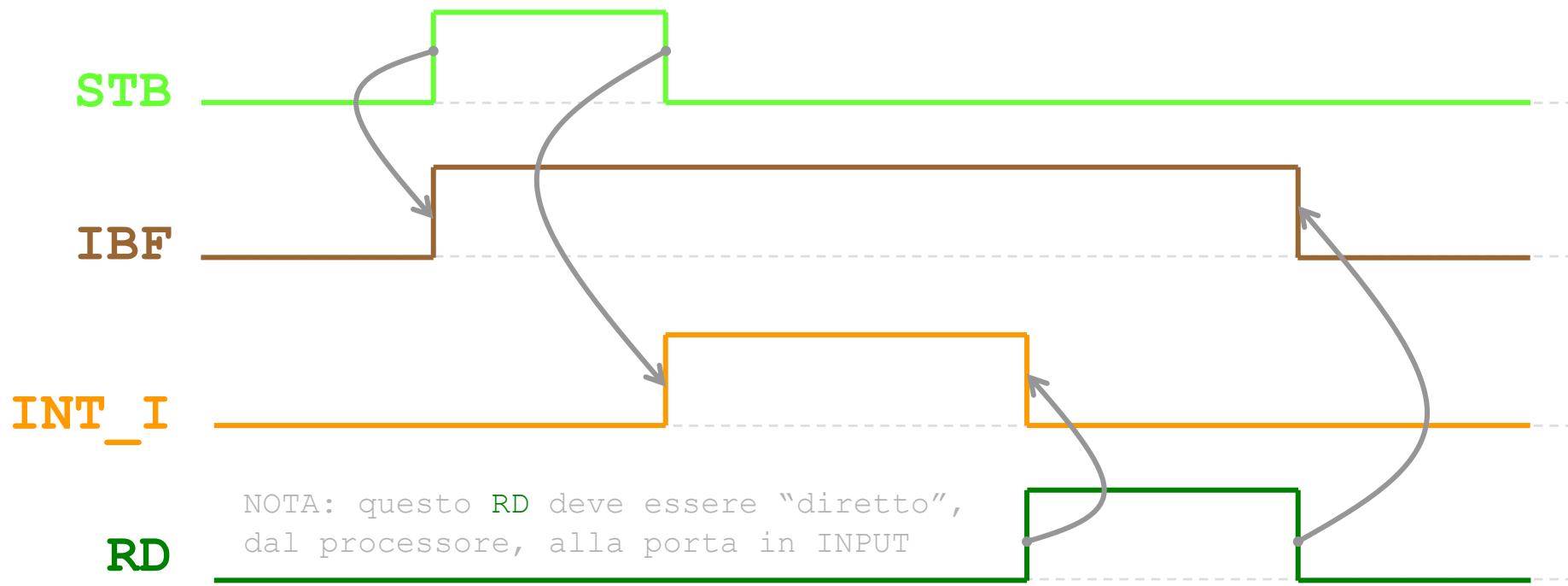
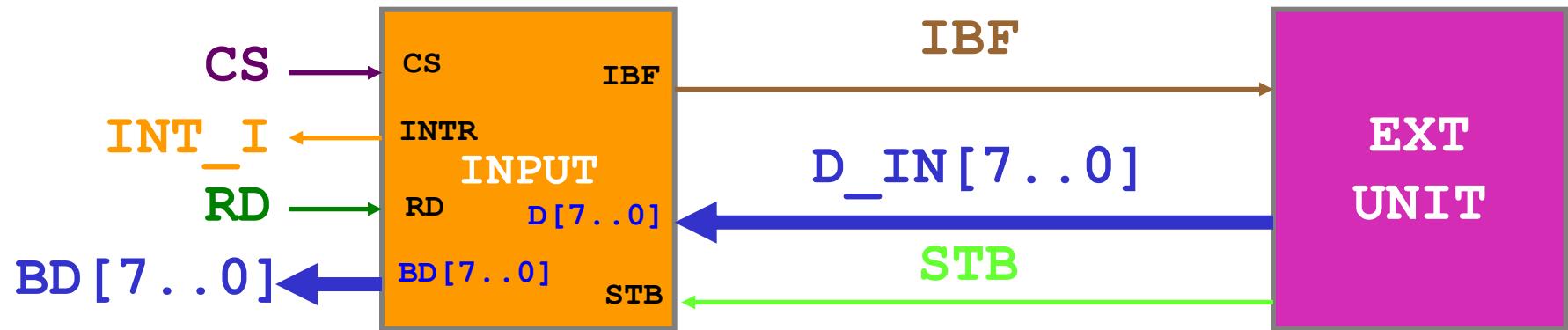
- Porta in **INPUT** mappata all'indirizzo **CS + 0**
- Porta in **OUTPUT** mappata all'indirizzo **CS + 1**

Al fine di risolvere il problema, risulta utile pensare la porta di I/O come **composta da due porte indipendenti**: una porta in **input** e una porta in **output**

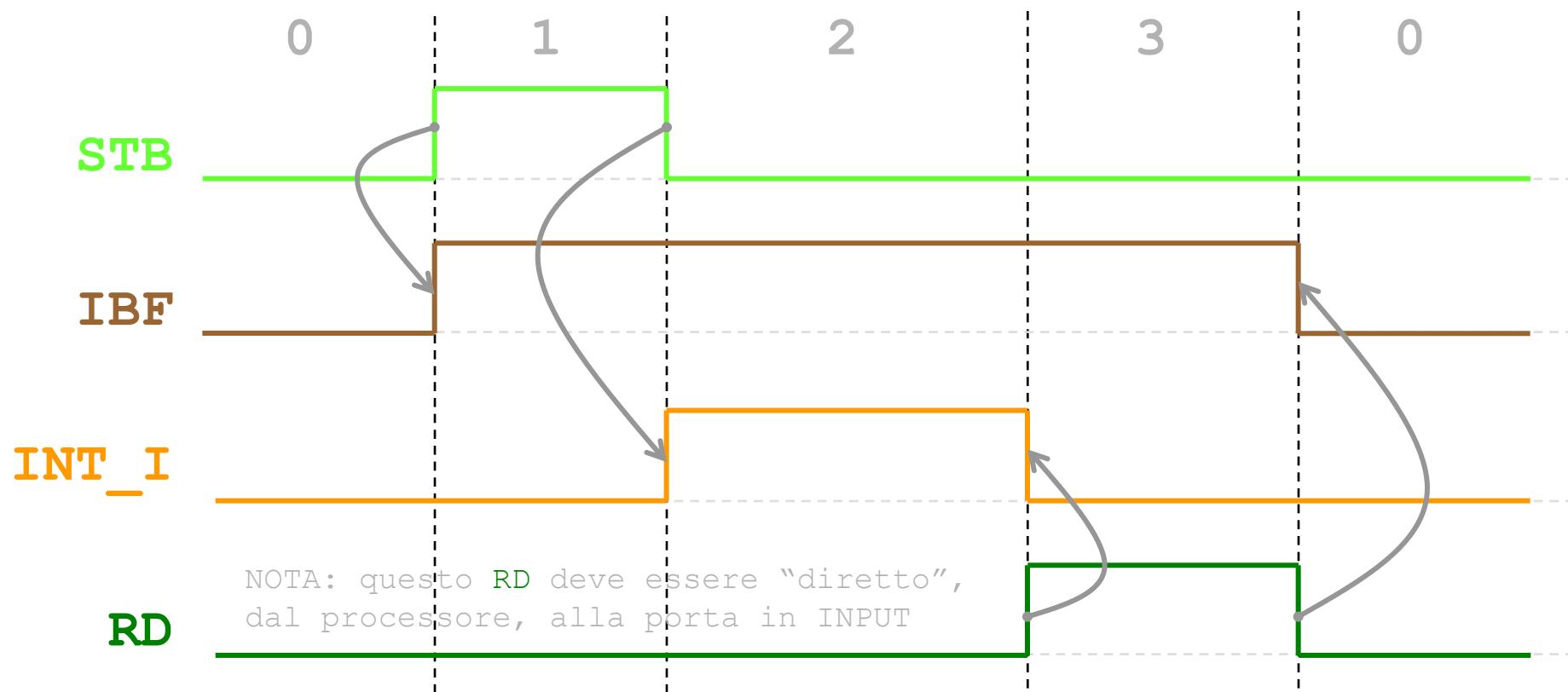
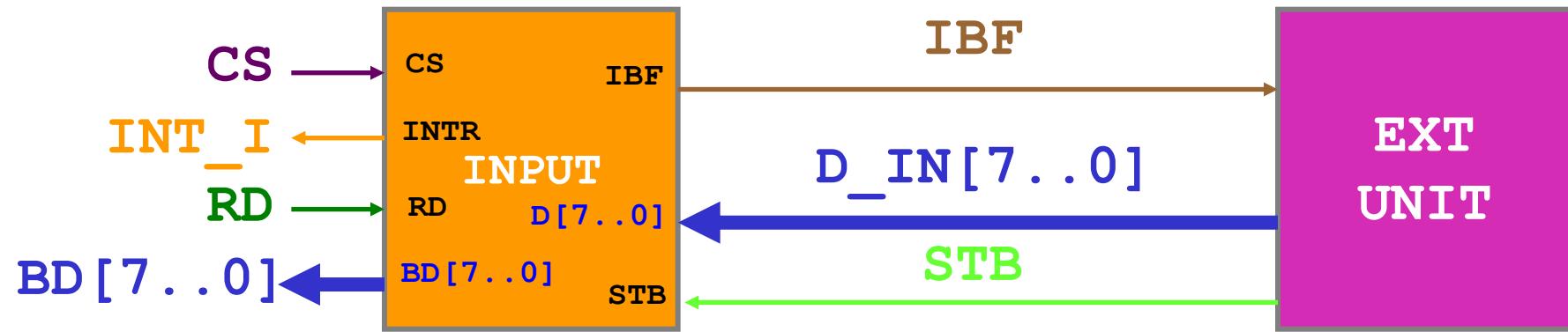
Inoltre, nella soluzione si desidera **evitare clock gating**

Il punto di partenza sono le **forme d'onda del protocollo di handshake**, mostrate nelle pagine precedenti

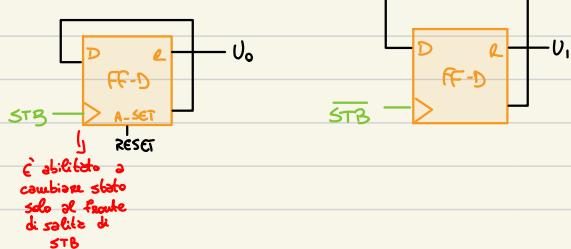
Handshake (INPUT)



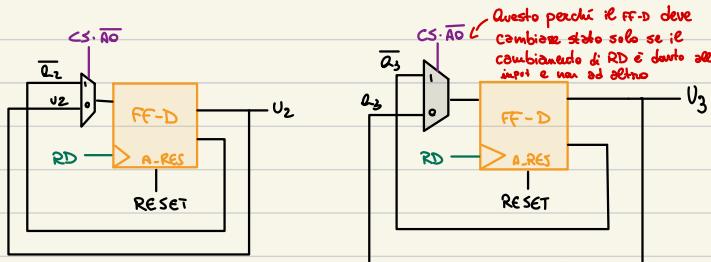
Handshake (INPUT)



Voglio monitorare i fronti di salita e discesa di STB e RD



La soluzione consiste nel controllare i fronti di salita e discesa di STB e RD in modo da individuare i diversi momenti del trasferimento e poi generare IBF e INT_1 come rate cambiatoria dei diversi momenti

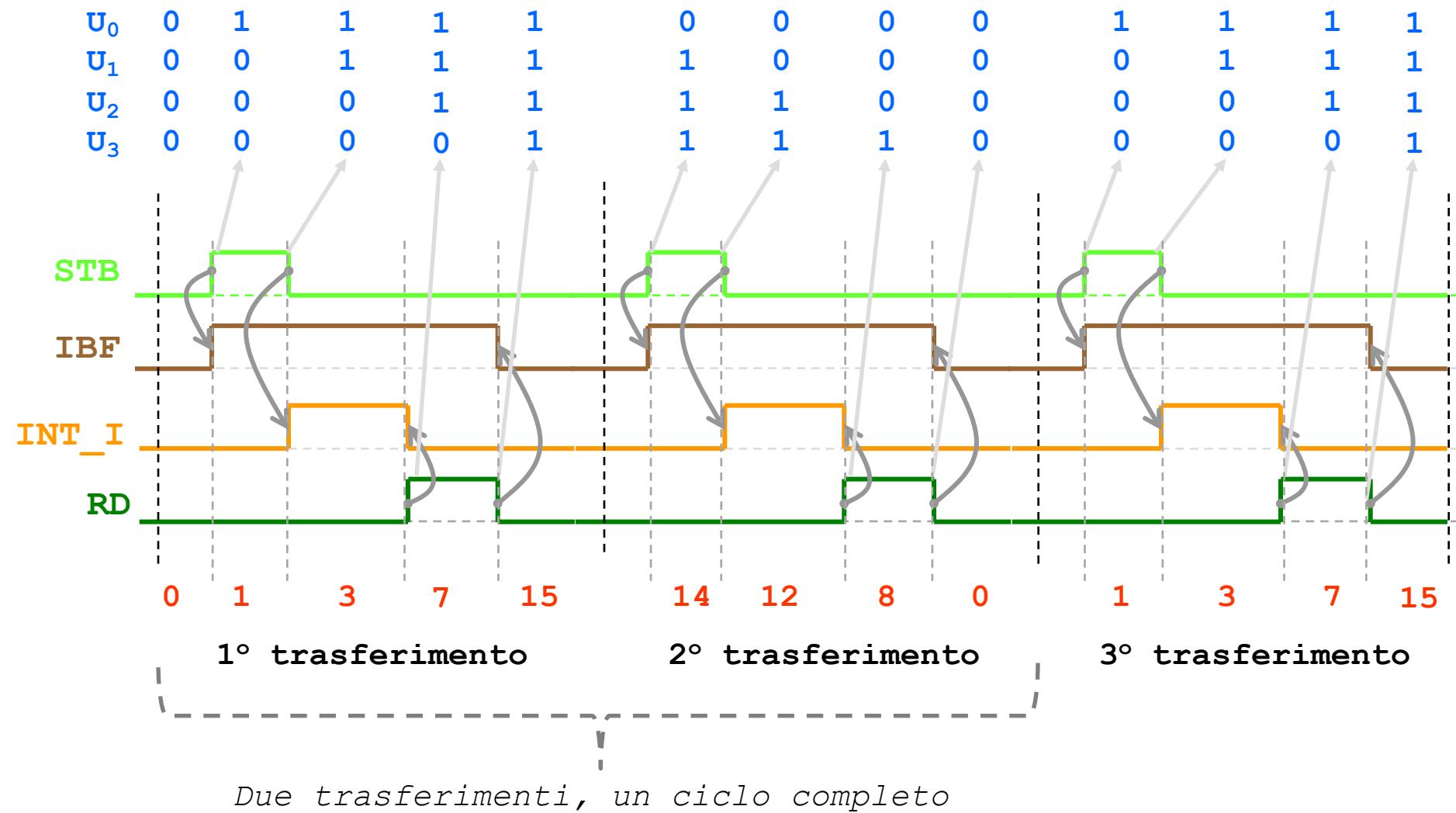


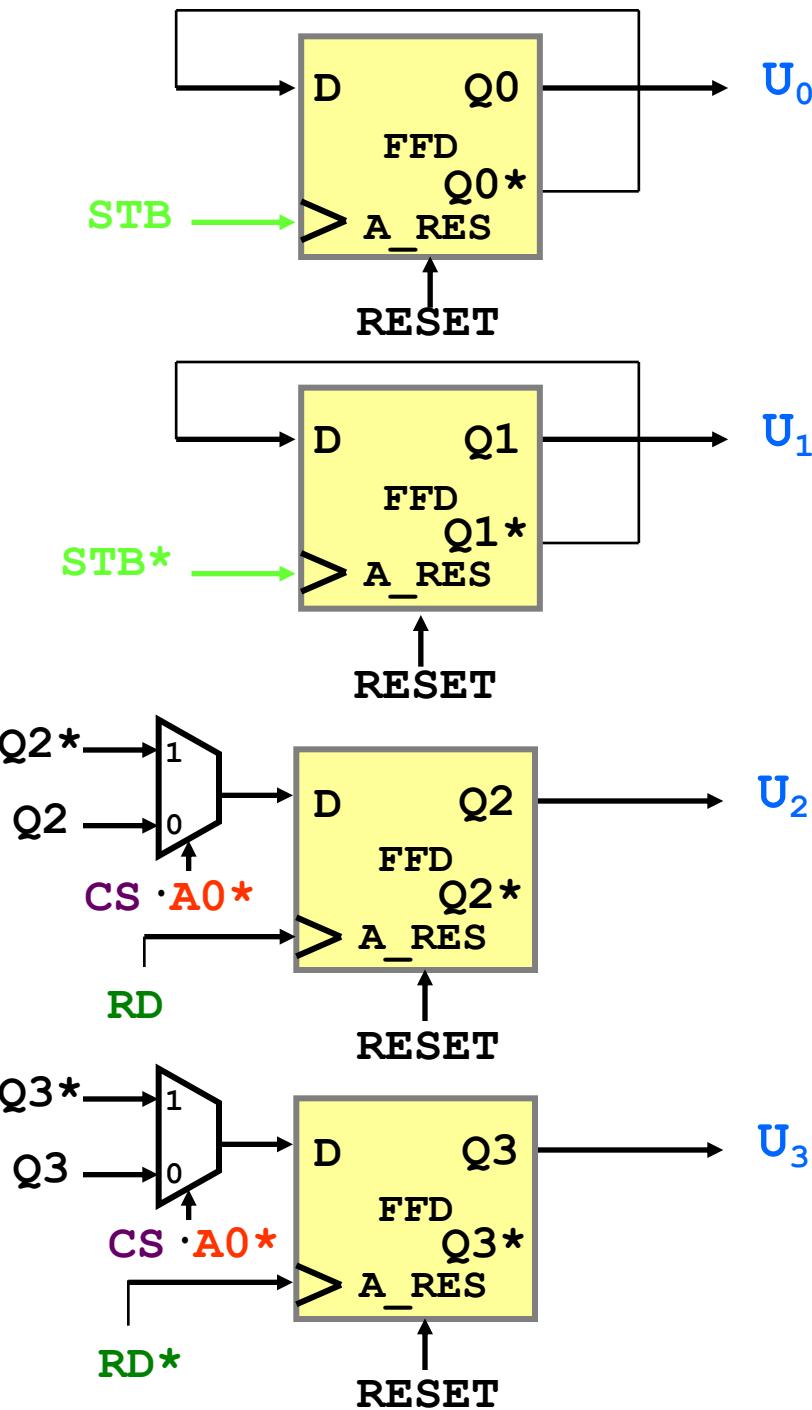
$$IBF = DEC(1) + DEC(3) + DEC(7) + DEC(14) + DEC(12) + DEC(18)$$

$$INT_1 = DEC(3) + DEC(12)$$

Handshake (INPUT): soluzione senza clock gating

Osservando le forme d'onda, è possibile individuare una soluzione senza *clock gating*





Il segnale $DEC(x)$ identifica la configurazione binaria $U_3 U_2 U_1 U_0$ equivalente a x in base 10. Pertanto, i segnali IBF e INT_I risultano:

$$\begin{aligned} IBF &= (DEC(1) + DEC(3) + DEC(7)) \\ &+ (DEC(14) + DEC(12) + DEC(8)) \end{aligned}$$

$$INT_I = DEC(3) + DEC(12)$$

Oppure,

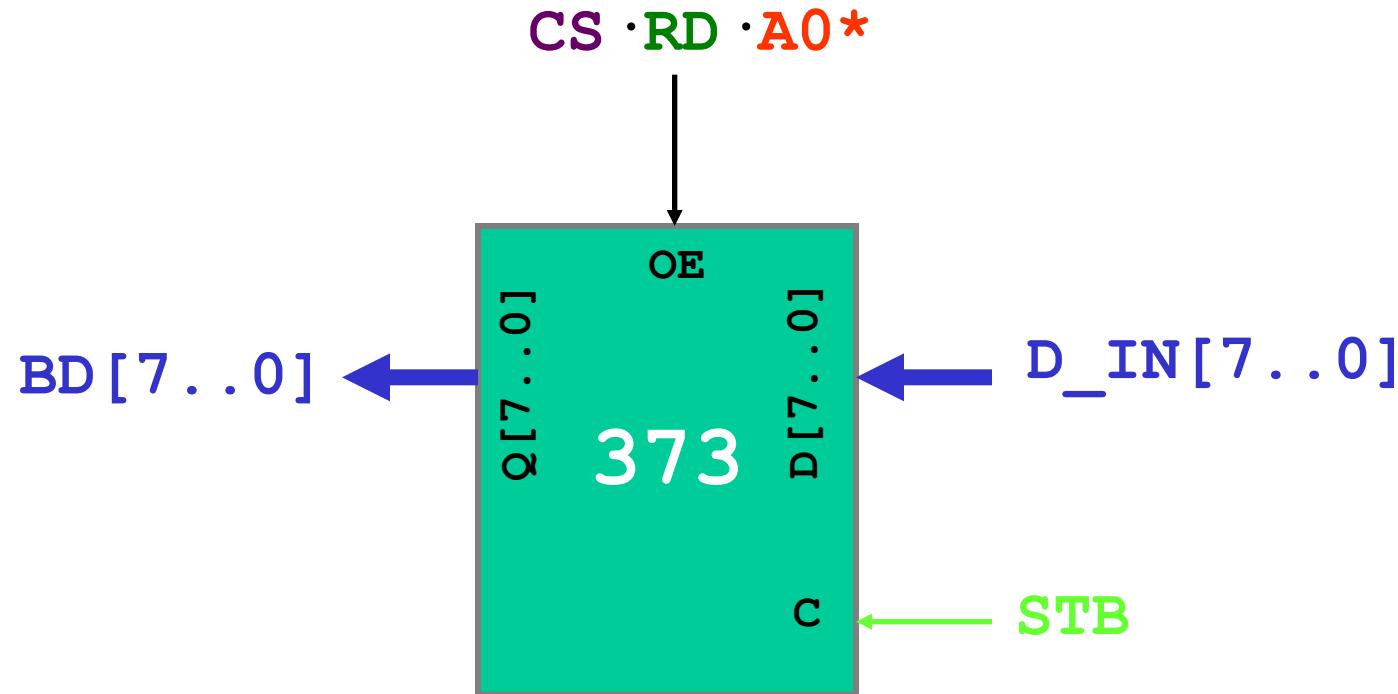
$$IBF = U_0 \text{ XOR } U_3$$

$$INT_I = U_1 \text{ XOR } U_2$$

CS è il *chip-select* della periferica in input

Handshake (INPUT) : buffer di ingresso con 373

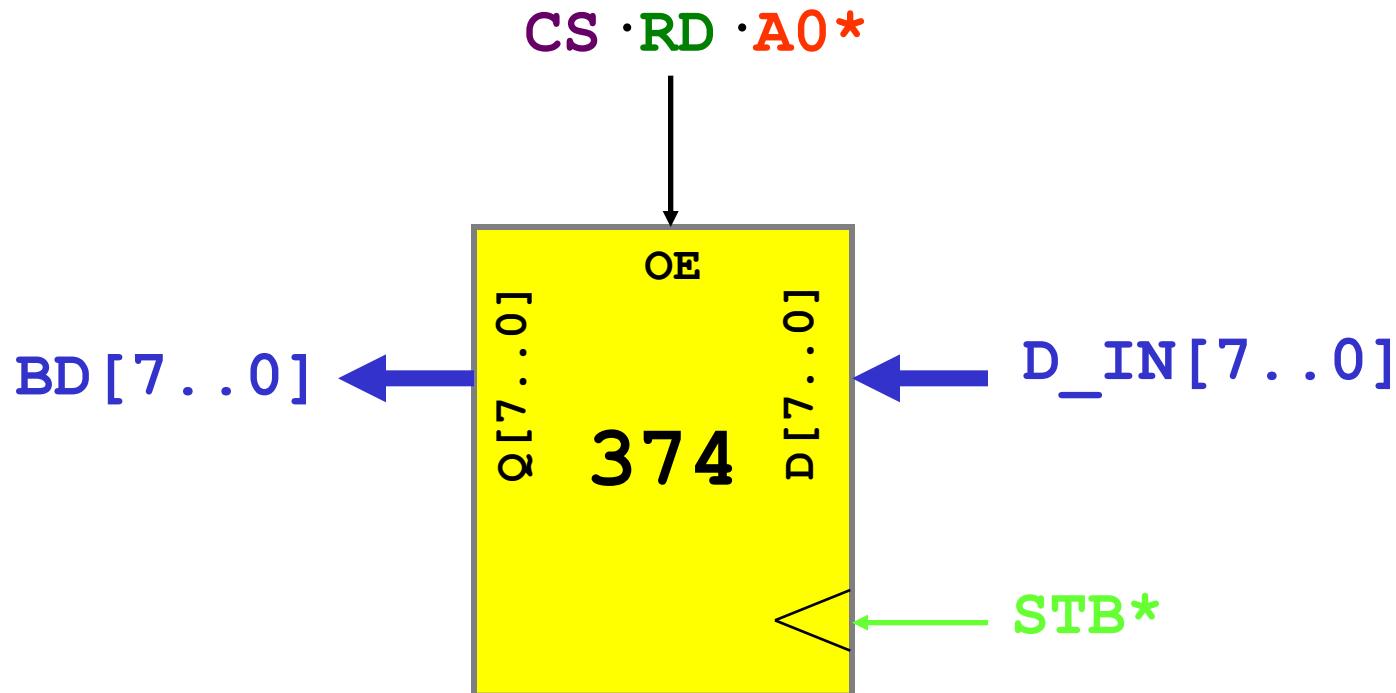
Ipotizzando di mappare la porta in **INPUT** all'indirizzo **CS + 0** e di voler utilizzare dei **latch 373 come buffer**.



Ovviamente sarebbe possibile una **soluzione del tutto equivalente con 374** come mostrato nella pagina successiva

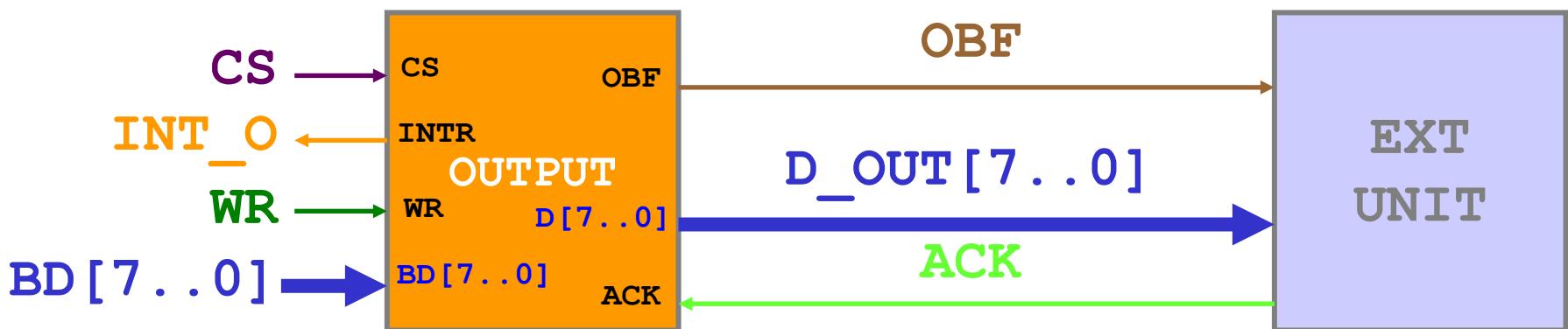
Handshake (INPUT) : buffer di ingresso con 374

Ipotizzando di mappare la porta in **INPUT** all'indirizzo **CS + 0** e di voler utilizzare dei **FFD 374** come **buffer**

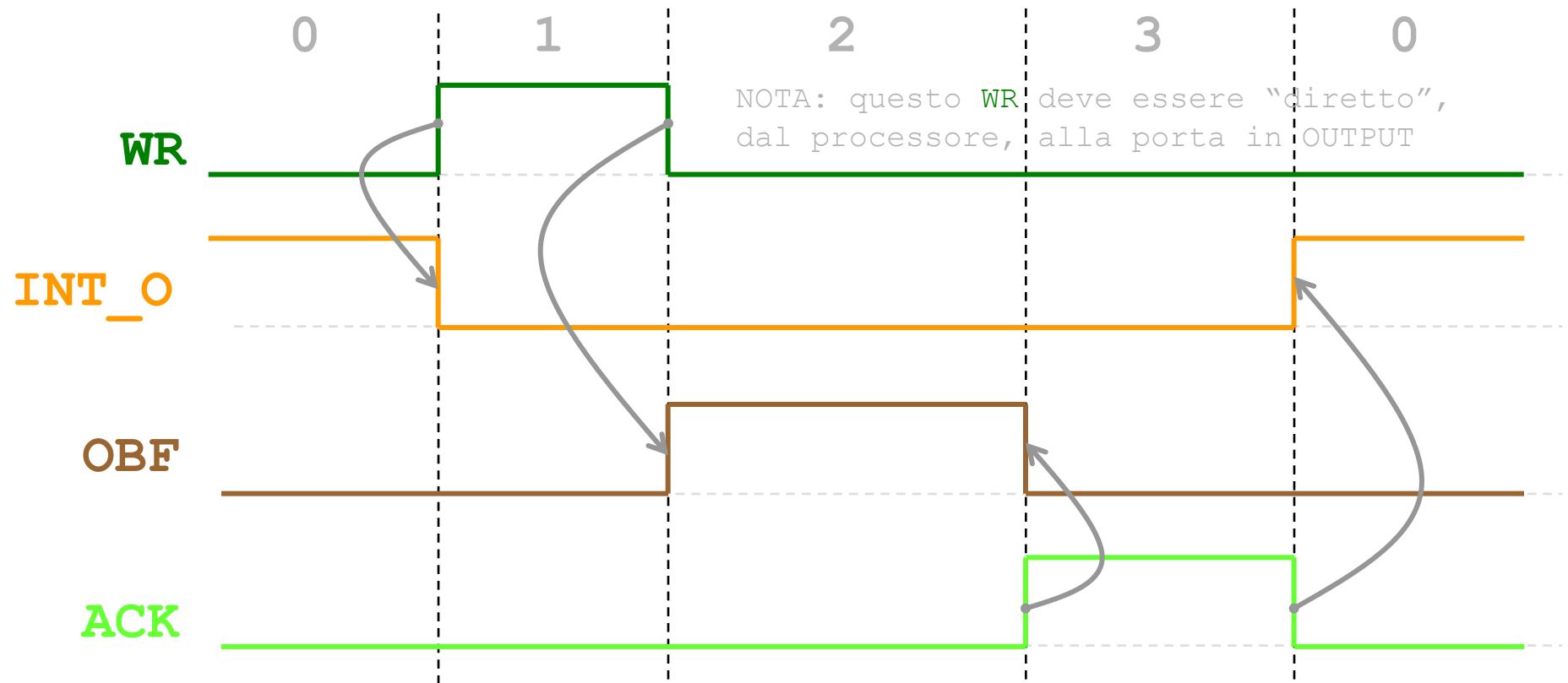
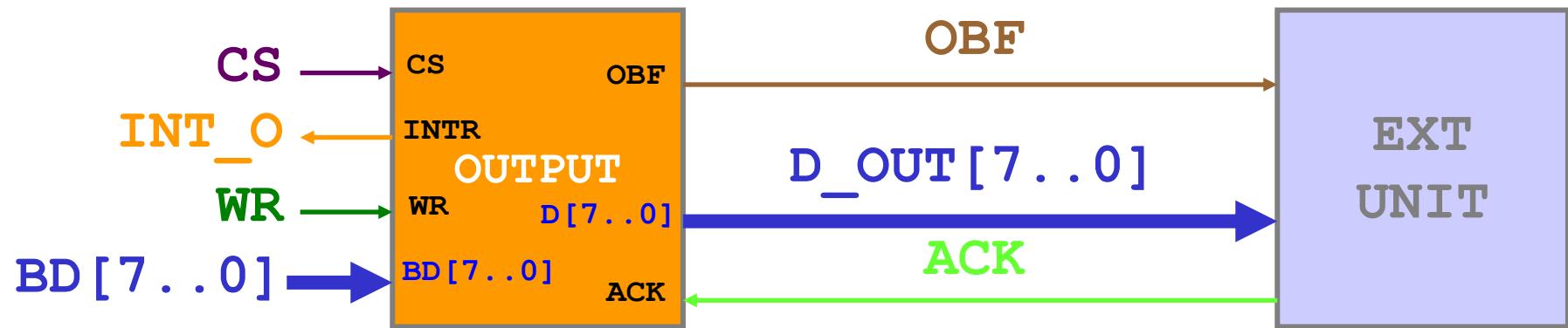


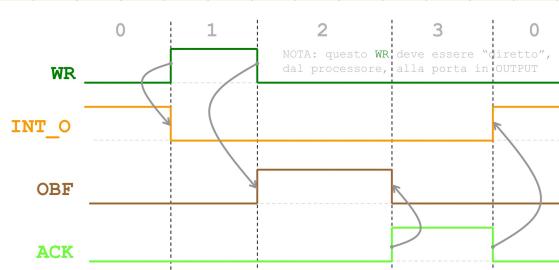
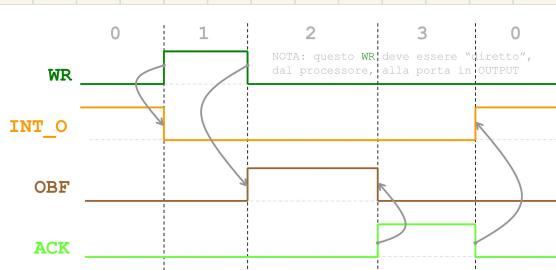
Esercizio: progetto della porta in output

Progettare la periferica per gestire i trasferimenti in Output mediante handshake a partire dalle forme d'onda mostrate nelle slide successive.



Handshake (OUTPUT)





Registri di stato e programmazione

Sarebbe utile aggiungere alla periferica che gestisce input e output con protocollo di handshake i seguenti registri:

- **Registro di stato** (lettura segnali di stato per gestione a polling) indirizzo **A1A0 = 10**
- **Registro di programmazione** (enable/disable singola interfaccia, etc) indirizzo **A1A0 = 11**

Ovviamente, serve un ulteriore bit (**A1**) per indirizzare gli ulteriori due registri

Esercizio

Come si potrebbe modificare il progetto della porta di I/O con handshake per aggiungere queste nuove funzionalità?

Esercizio

Progettare un sistema basato sul microprocessore **DLX**, con **1 GB di EPROM agli indirizzi bassi e 2 GB di RAM agli indirizzi alti**. Nel sistema è presente una porta in input, già progettata e denominata **INPUT_PORT**, e un pulsante denominato **P**. Il byte (unsigned) **letto da INPUT_PORT deve essere memorizzato** all'indirizzo **FFFF0020h** mentre il registro **R20 deve essere incrementato di uno**, via software, **a ogni pressione di P** e **inizializzato a 0 all'avvio** del sistema. Inoltre, si assuma che:

- 1) Il pulsante P abbia priorità maggiore di INPUT_PORT
- 2) Il pulsante P non possa essere premuto prima che sia terminata la gestione di P da parte dell'interrupt handler. A tal fine segnalare, con un LED, quando il pulsante non deve essere premuto
- 3) I registri da R25 a R30 possono essere utilizzati senza la necessità di essere ripristinati
- 4) Il registro R20 sia modificabile solo dall'handler che gestisce il pulsante

1 GB EPROM	INPUT	ANVIO E INTERRUPT	2GB RAM
------------	-------	----------------------	---------

0x4000

0x6000

0x8000

CS_EPROM_0 = BA31 BA30 BE0

CS_EPROM_1 = " BE1

CS_EPROM_2 = " BE2

CS_EPROM_3 = " BE3

CS_RAM_0 = BA31 BE0

CS_RAM_1 = " BE1

CS_RAM_2 = " BE2

CS_RAM_3 = " BE3

CS_INPUT_READ = BA31 BA30 BA29 0x4000

CS.LEGGI_ANVIO = BA31 BA30 BA29 BA3 BA2 MEMRD 0x6000

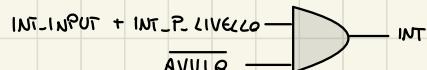
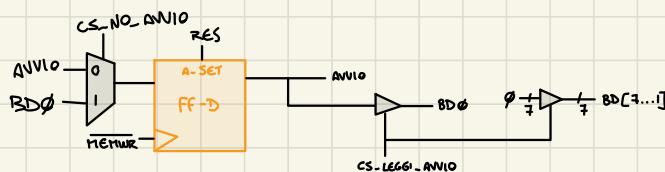
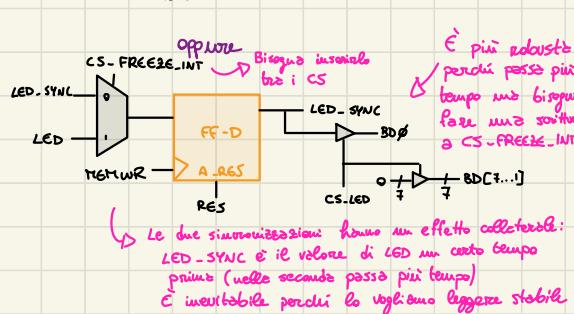
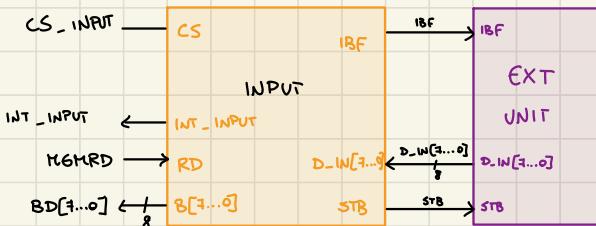
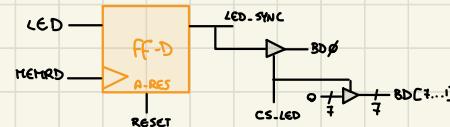
CS_NO_ANVIO = BA31 BA30 BA29 BA3 BA2 0x6000 00004

CS_INT_P_0 = BA31 BA30 BA29 BA3 BA2 MEMWR 0x6000 00008

CS_LED = BA31 BA30 BA29 BA3 BA2 MEMRD 0x6000 000C

Potevo usare
anche i BE

Dovrò sincronizzare
LED perché potrai
leggere mentre
sta cambiando
(in metastabilità)



CODICE

init: LHI R25, 0x6000;

LBU R30, 0x0000(R25);

BEQZ R30, handle;

LHI R20, 0x0000, // imposto R20
// a 0 (solo all' avvio)SB R0, 0x0004(R25); // netto avvio
// a 0

J main;

handler: LBU R30, 0x000C(R25);

BEQZ R30, input; // se LED = 0 allora è
// INPUT

ADDI R20, R20, 0x0001; // R20 di 1

SB R0, 0x0008(R25); // imposto INT_P
// a 0

RFE;

init: LHI R25, 0x4000; // solo in
// R30 il valore

LBU R30, 0x0000(R25); // di input

LHI R25, 0xFFFF; // solo in
// R30 per ilSB R30, 0x0020(R25); // contenuto di R30
// ovvero ciò cheho letto dalle
piste in input

main: